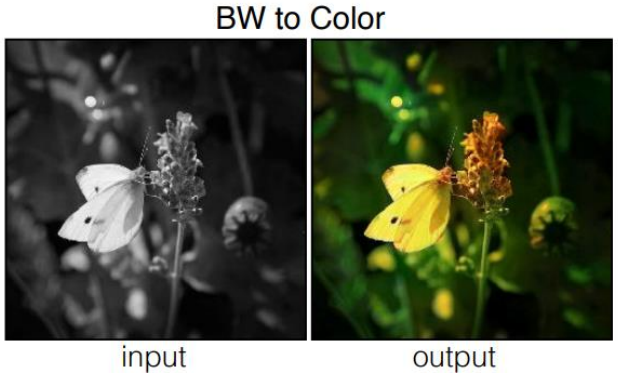
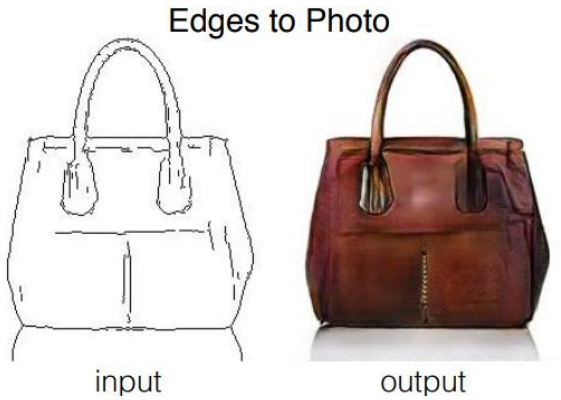
Image-to-Image Translation with Conditional Adversarial Networks

Diego Calabretta 1000012346

Group J - a.a. 2020-21

Most of problems in image processing and computer vision can be considered as translating an input image into a corresponding output image





So far these kind of problem was dealing with special-purpose approach, even if the processing task is very similar in each applications.

The paper aim to develop a general-purpose solution with Generative Adversarial Networks. GANs learn a loss that tries to classify if the output image is real or fake, while simultaneously training a generative model to minimize this loss. In particular, we will use conditional GANs (cGANs) that learn a conditional generative model.

Method

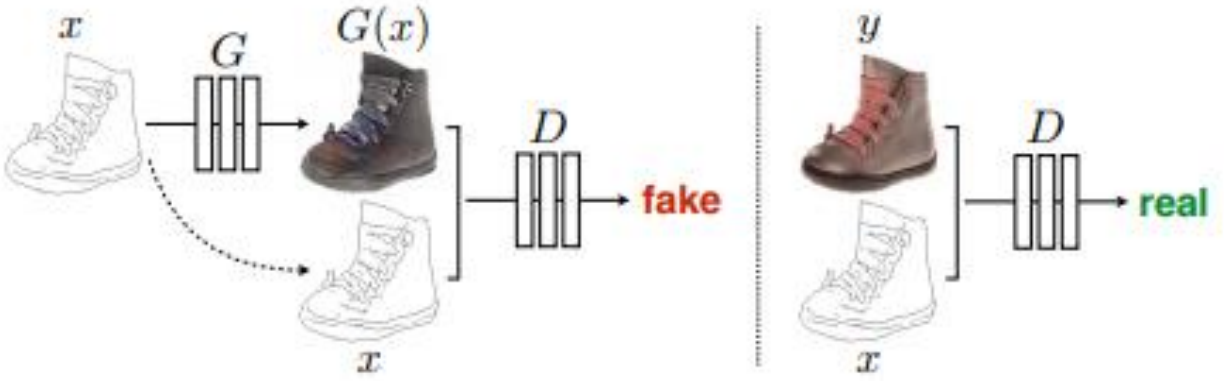
GANs are generative models that learn a mapping from random noise vector z to output image y,

$G : z \rightarrow y$

cGANs learn a mapping from observed image x and random noise vector z, to y,

$G : \{x, z\} \rightarrow y.$

The **discriminator**, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The **generator**, G, learns to fool the discriminator.



Unconditional GAN: $$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_y[\log D(y)] + \mathbb{E}_{x,z}[\log(1 - D(G(x, z))].$$

Conditional GAN: $$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))]$$

L1 distance: $$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1].$$

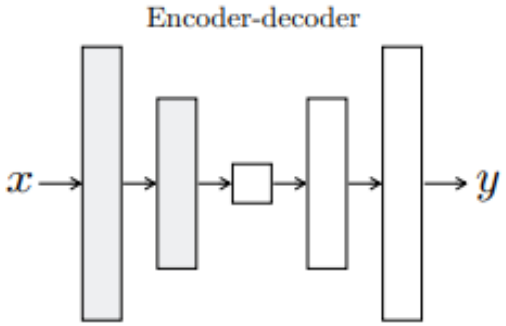
Final objective: $$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

The final models provide noise (z) only in the form of dropout, applied on several layers of our generator at both training and test time. It was observed only minor stochasticity in the output of our nets.

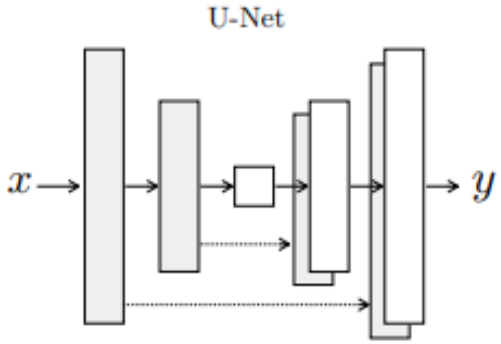
Architettura

For the problems we consider, the input and output differ in surface appearance, but both are renderings of the same underlying structure.

Previous solutions to problems in this area have used an encoder-decoder network where the input is passed through a series of layers that progressively downsample, until a bottleneck layer, at which point the process is reversed.
This requires that all information flow pass through all the layers, but in many image translation problems, we can shuttle some low-level information directly across the net.



The generator of the paper is a U-Net with skip connections between each layer i and layer n − i, where n is the total number of layers. Each skip connection simply concatenates all channels at layer i with those at layer n − i.



The GAN discriminator we use is restricted to only model high-frequency structure, because low-frequencies are already captured by L1 loss.

Therefore is sufficient to design a **PatchGAN** that give attention to the structure in local images patch.

This discriminator tries to classify if each N ×N patch in an image is real or fake. We run this discriminator convolutionally across the image, averaging all responses to provide the ultimate output of D. N can be much smaller than the full size of the image.

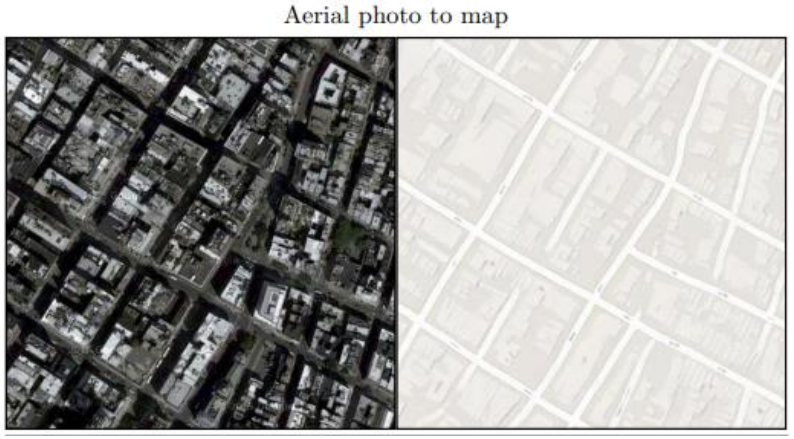
Such a discriminator effectively models the image as a Markov random field, assuming independence between pixels separated by more than a patch diameter. (common assumption in models of texture and style)

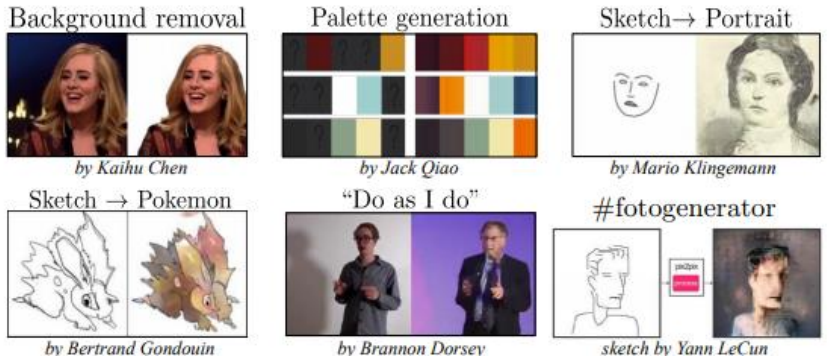
Optimization

- Alternate between one gradient descent step on D, then one step on G. Rather than training G to minimize $\log(1 - D(x, G(x, z))$, we instead train to maximize $\log D(x, G(x, z))$
- Divide the objective by 2 while optimizing D, which slows down the rate at which D learns relative to G.
- Use minibatch SGD and apply the Adam solver, with a learning rate of 0.0002, and momentum parameters $\beta 1 = 0.5$, $\beta 2 = 0.999$.
- Run the generator net in exactly the same manner as during the training phase and inference time. This differs from the usual protocol in that we apply dropout at test time, and we apply batch normalization using the statistics of the training batch.

Experiments

- Semantic labels↔photo
- Architectural labels→photo
- Map↔aerial photo
- BW→color photos
- Edges→photo
- Sketch→photo
- Day→night
- Thermal→color photos
- Photo with missing pixels→inpainted photo





Evaluation metrics

- **Amazon Mechanical Turk (AMT):** AMT was used **f**or graphics problems in map generation, aerial photo generation, and image colorization. It were presented “real” image against a “fake” image generated by paper’s algorithm (10 images with feedback, 40 images with no feedback, appeared for 1 second and unlimited time to respond).

- **FCN-score:** FCN-8S, is a fully convolutional networks for semantic segmentation, trained on the cityscapes dataset. They present the synthesized photos by the classification accuracy against the labels these photos were synthesized from. If the generated images are realistic, classifiers will be able to classify the synthesized image correctly as well.

- **Objective function**

In the figure we can see the effects of L1,GAN,cGAN term:

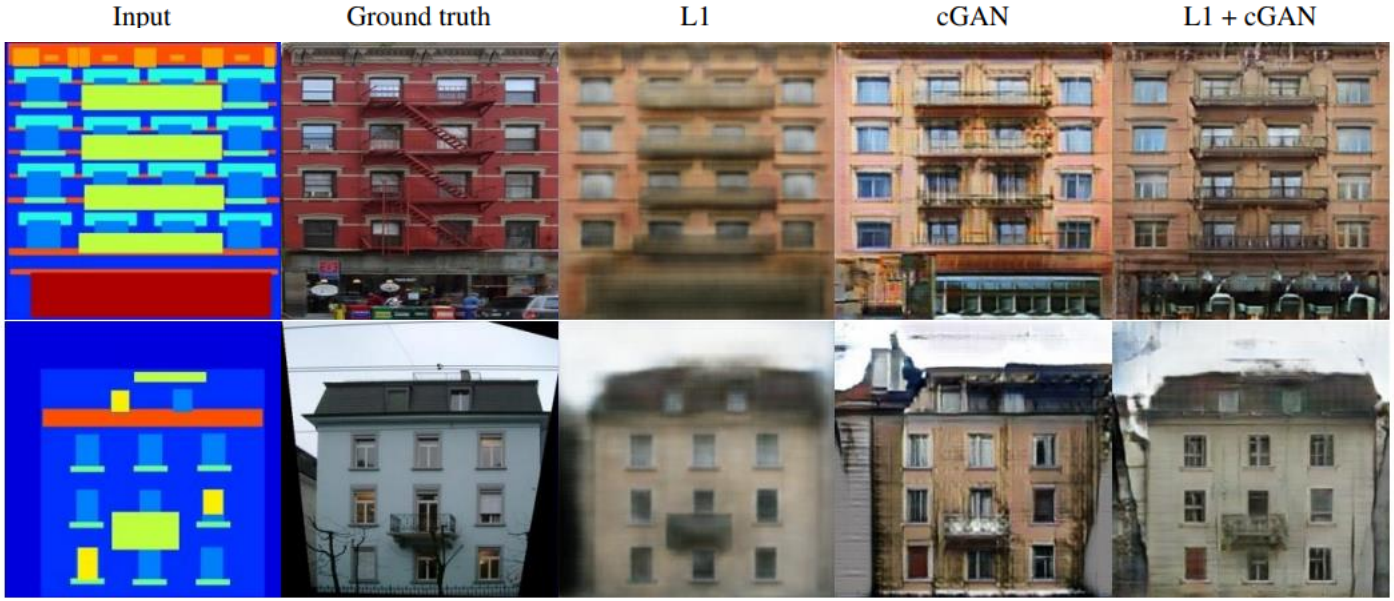


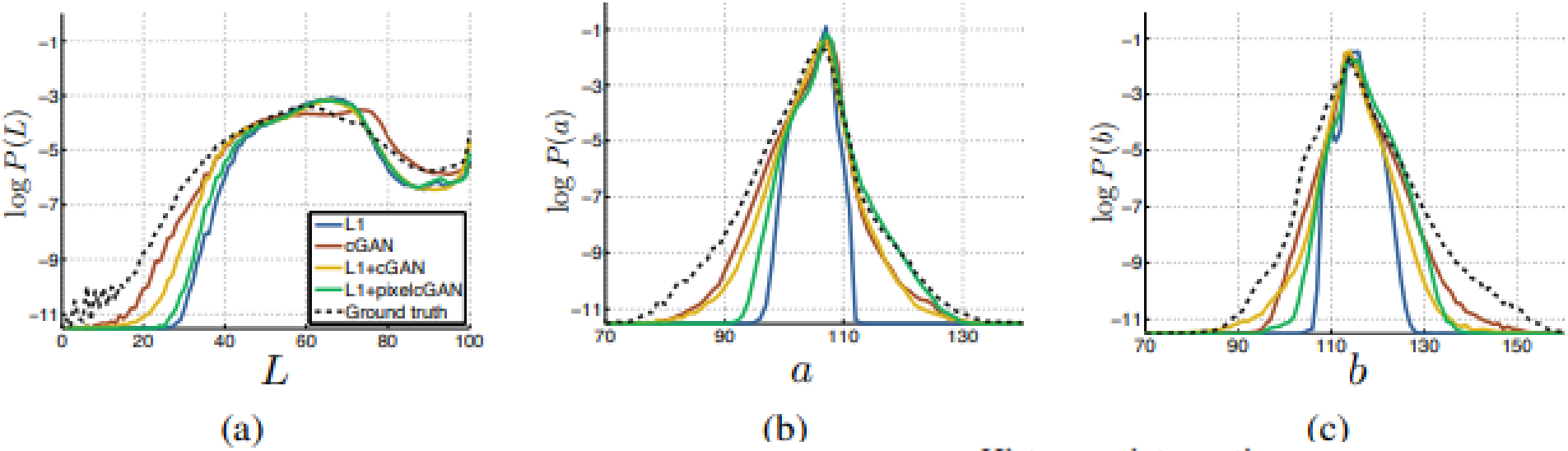
Table show results of FCN-score on the cityscapes application:

Loss	Per-pixel acc.	Per-class acc.	Class IOU
L1	0.42	0.15	0.11
GAN	0.22	0.05	0.01
cGAN	0.57	0.22	0.16
L1+GAN	0.64	0.20	0.15
L1+cGAN	**0.66**	**0.23**	**0.17**
Ground truth	0.80	0.26	0.21

- **L1** give a blurred image.
- **cGAN** give a good image but with visual artifact.
- Using **L1 + cGAN** the result is very accurate and realistic

- **Colors**

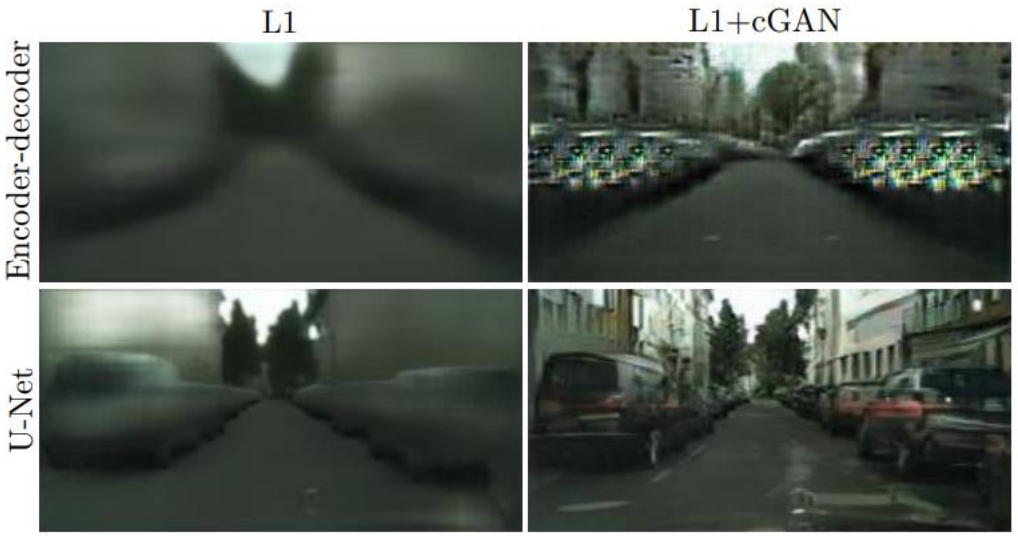
  L1 will incentivize a blur when it is uncertain where exactly to locate an edge, it will also incentivize an average, grayish color when it is uncertain which of several plausible color (unrealistic). On the other hand, GANs produce sharp images, hallucinating spatial structure even where it does not exist in the input label map, making images more colorful.



The graphs show color distribution matching property of the cGAN

Loss	Histogram intersection against ground truth L	a	b
L1	0.81	0.69	0.70
cGAN	**0.87**	0.74	**0.84**
L1+cGAN	0.86	**0.84**	0.82
PixelGAN	0.83	0.68	0.78

- **Encoder-decoder vs U-Net architecture**



Loss	Per-pixel acc.	Per-class acc.	Class IOU
Encoder-decoder (L1)	0.35	0.12	0.08
Encoder-decoder (L1+cGAN)	0.29	0.09	0.05
U-net (L1)	0.48	0.18	0.13
U-net (L1+cGAN)	**0.55**	**0.20**	**0.14**

- **From PixelGANs to PatchGANs to ImageGANs**

Test varying the patch size N of our discriminator receptive fields, from a 1 × 1 (PixelGAN), to 16x16 and 70x70 (PatchGAN), to 286x286 (ImageGAN)

Discriminator receptive field	Per-pixel acc.	Per-class acc.	Class IOU
1×1	0.39	0.15	0.10
16×16	0.65	0.21	**0.17**
70×70	**0.66**	**0.23**	**0.17**
286×286	0.42	0.16	0.11

- **Perceptual validation**

  Test with AMT

Loss	Photo → Map % Turkers labeled *real*	Map → Photo % Turkers labeled *real*
L1	2.8% ± 1.0%	0.8% ± 0.3%
L1+cGAN	6.1% ± 1.3%	**18.9% ± 2.5%**

Map to aerial photo

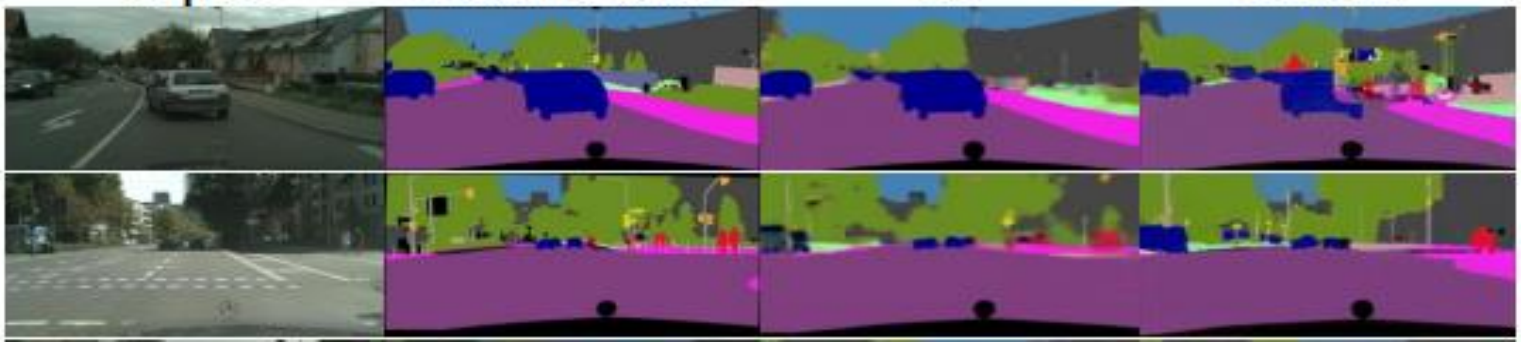
Aerial photo to map

- **Semantic Segmentation**

  Test where output is instead less complex than the input

Loss	Per-pixel acc.	Per-class acc.	Class IOU
L1	**0.86**	**0.42**	**0.35**
cGAN	0.74	0.28	0.22
L1+cGAN	0.83	0.36	0.29

Input Ground truth L1 cGAN



Conclusion

These networks learn a loss adapted to the task and data at hand, which makes them applicable in a wide variety of settings. Twitter community proved this thesis, applying pix2pix in many tasks far beyond the scope of the original paper.

Learning to see: Gloomy Sunday - YouTube

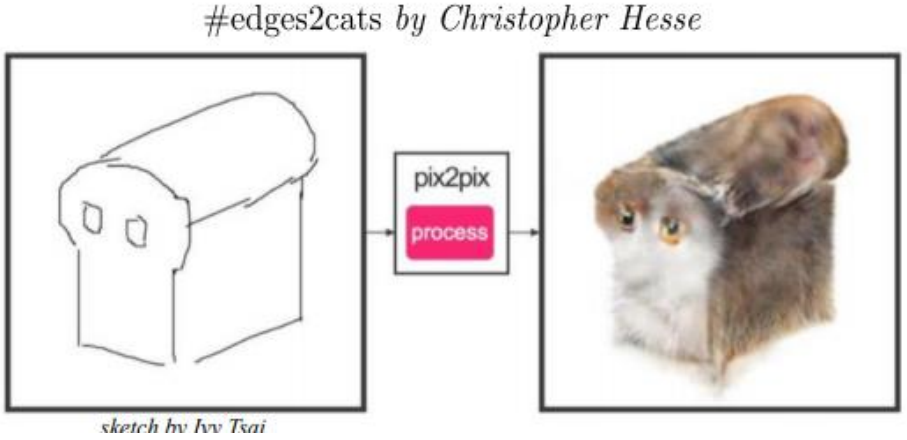
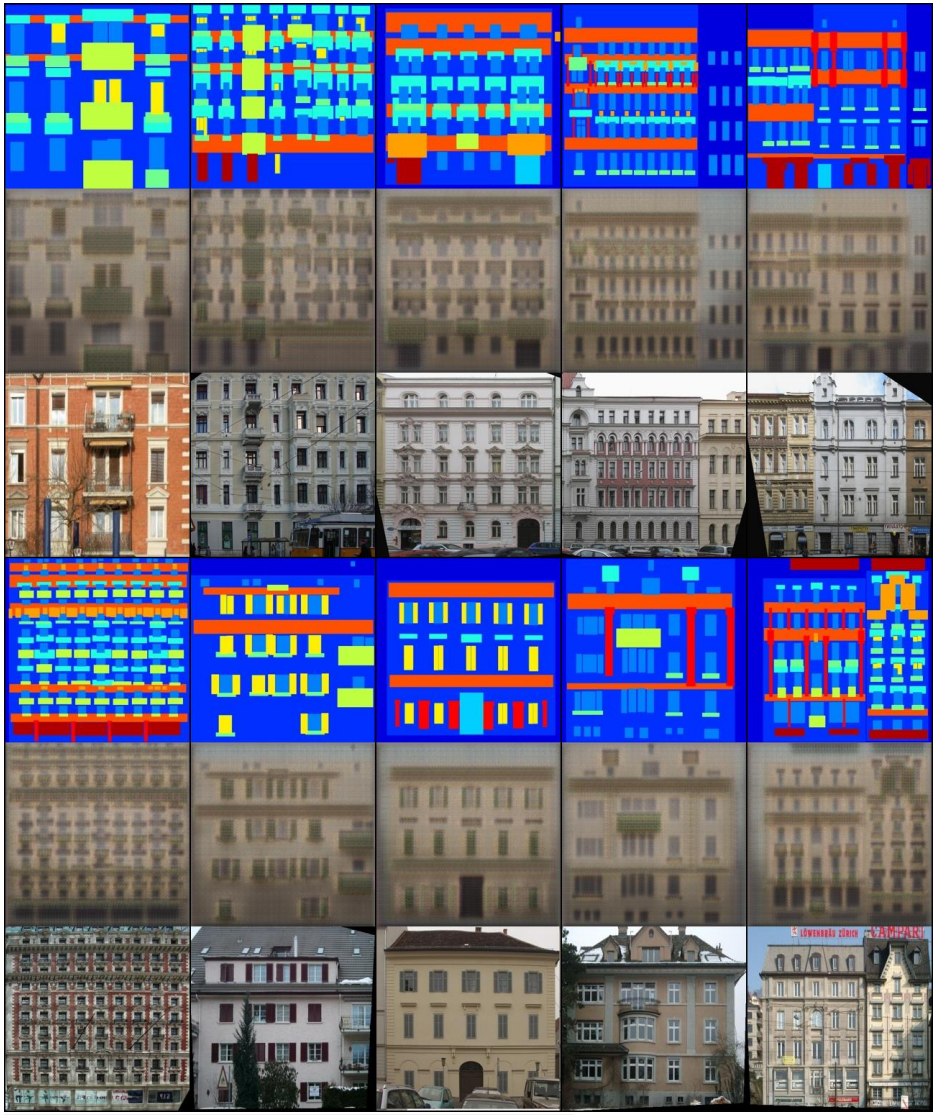
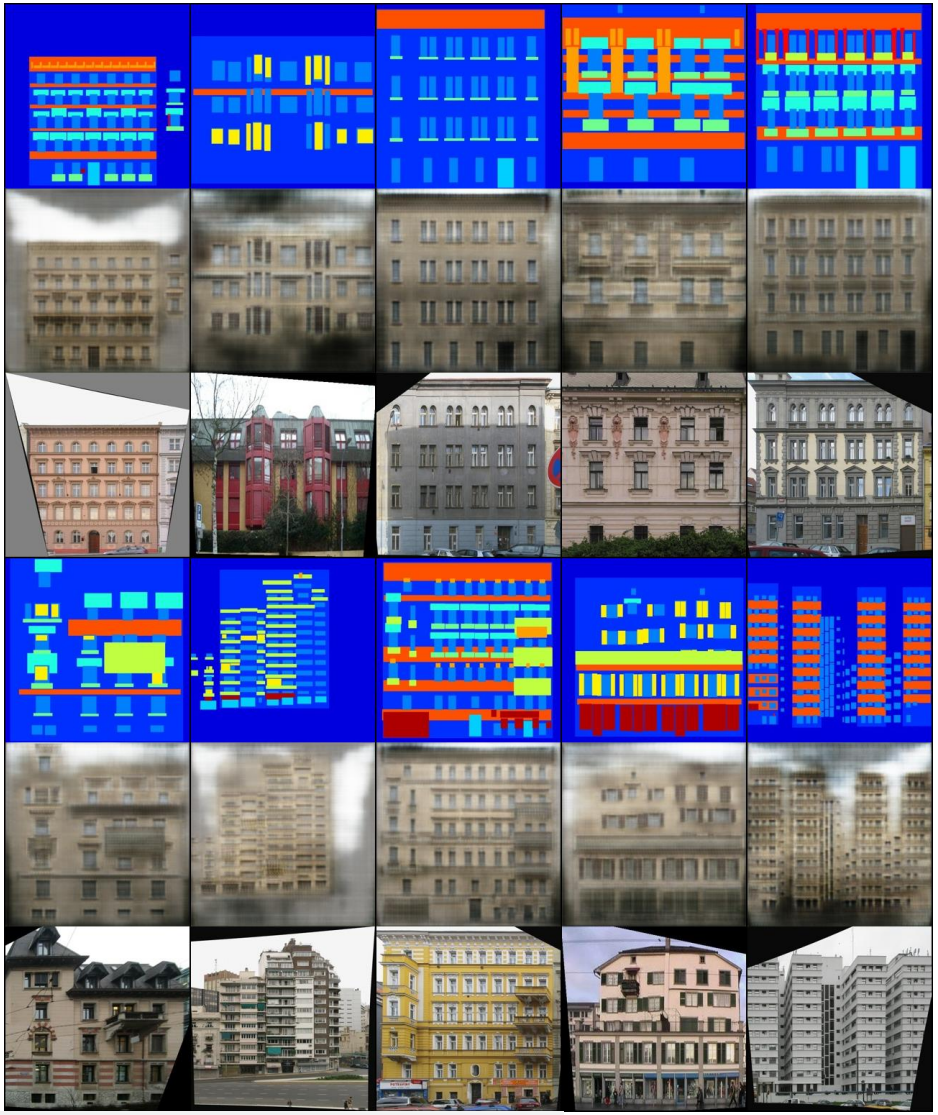


Image-to-Image Demo - Affine Layer

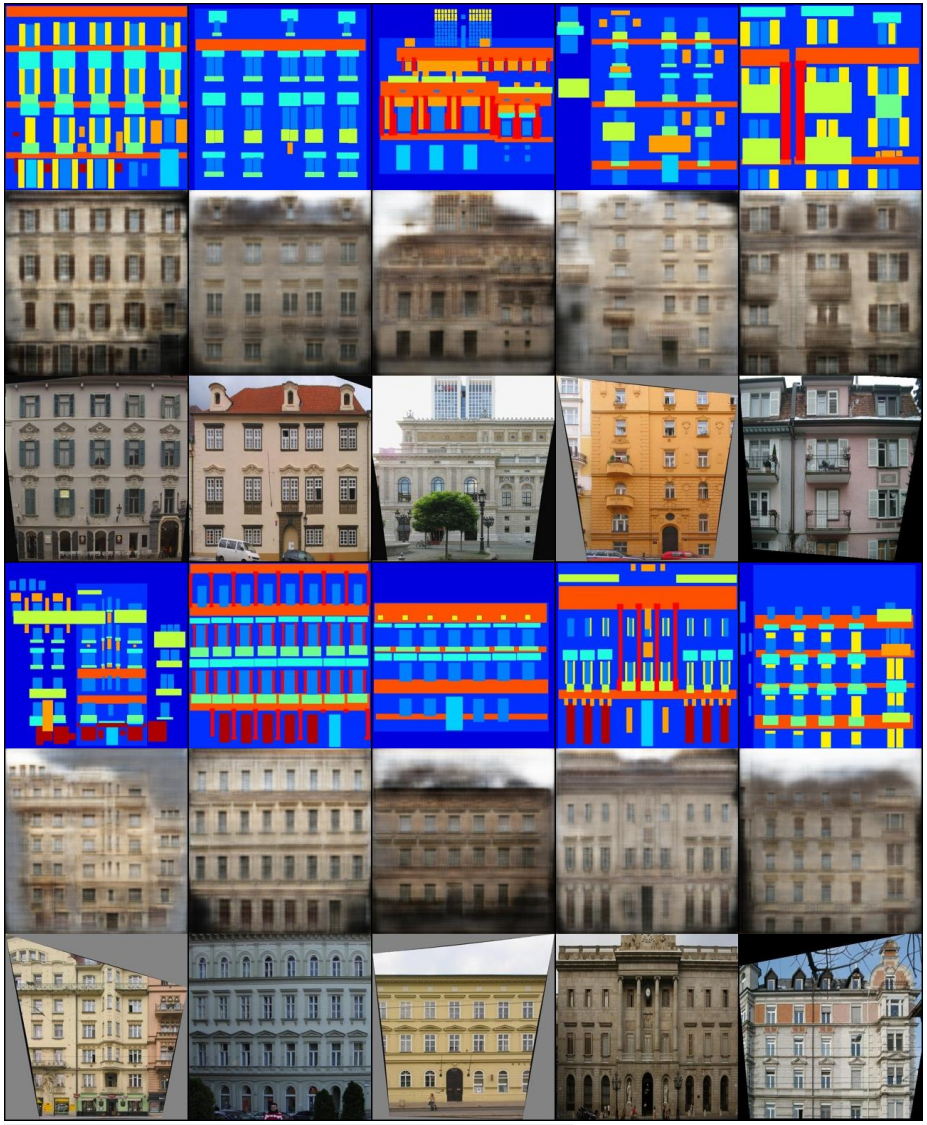
My results on Facades dataset



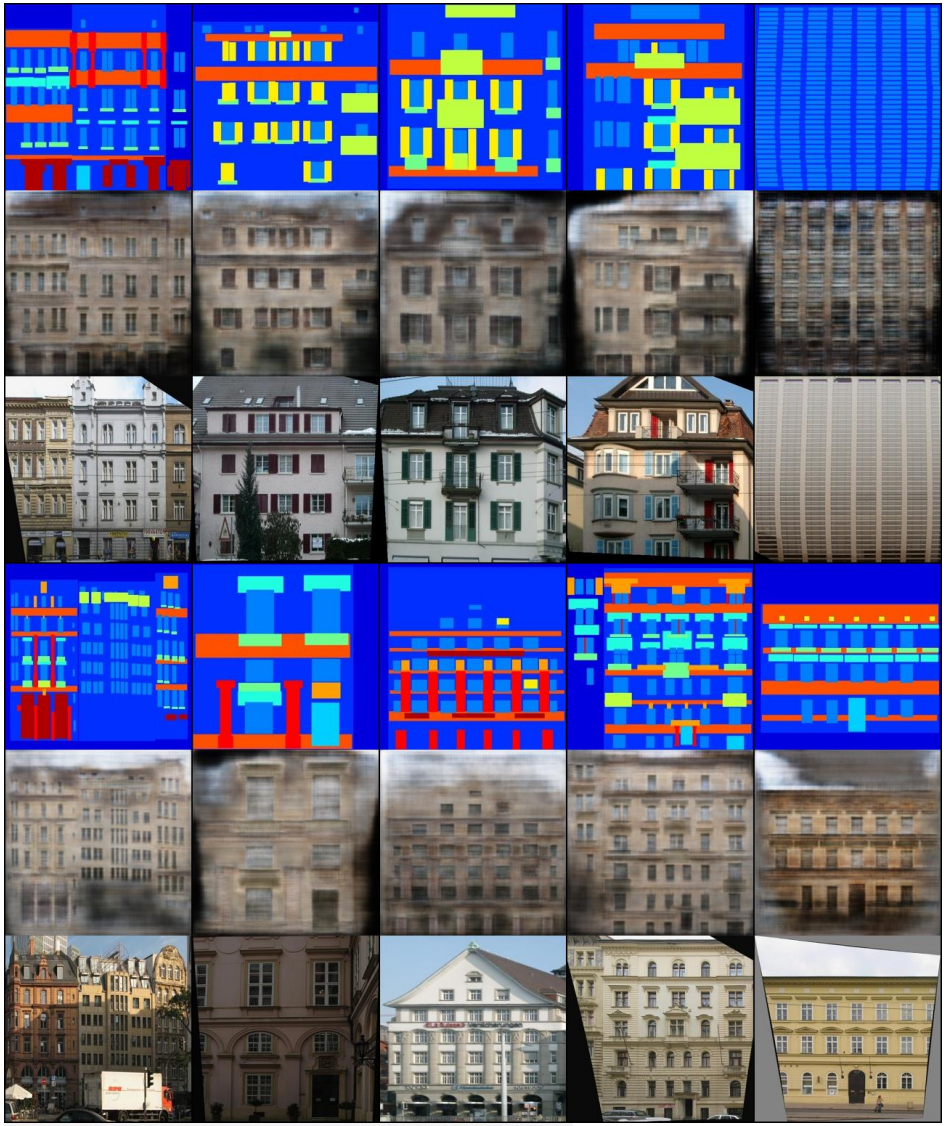
After 500 iteractions



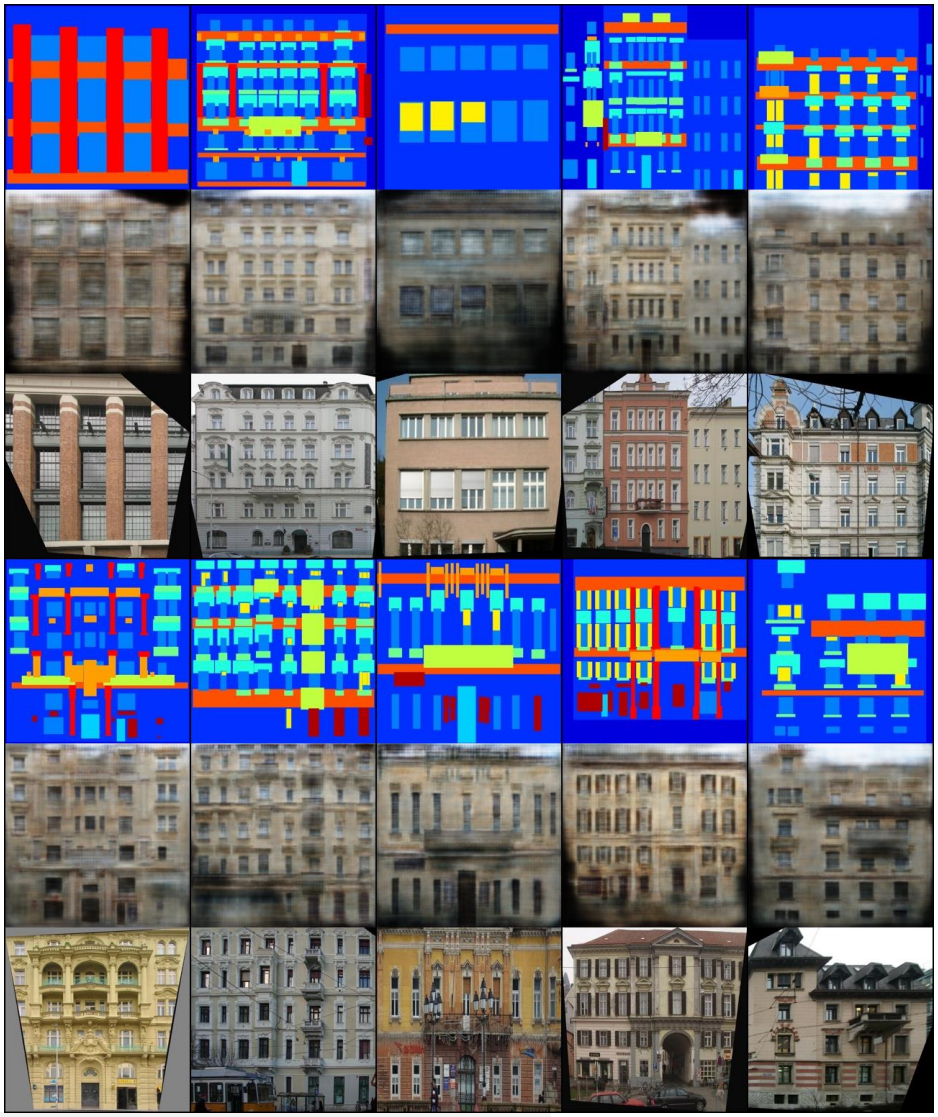
After 12500 iteractions



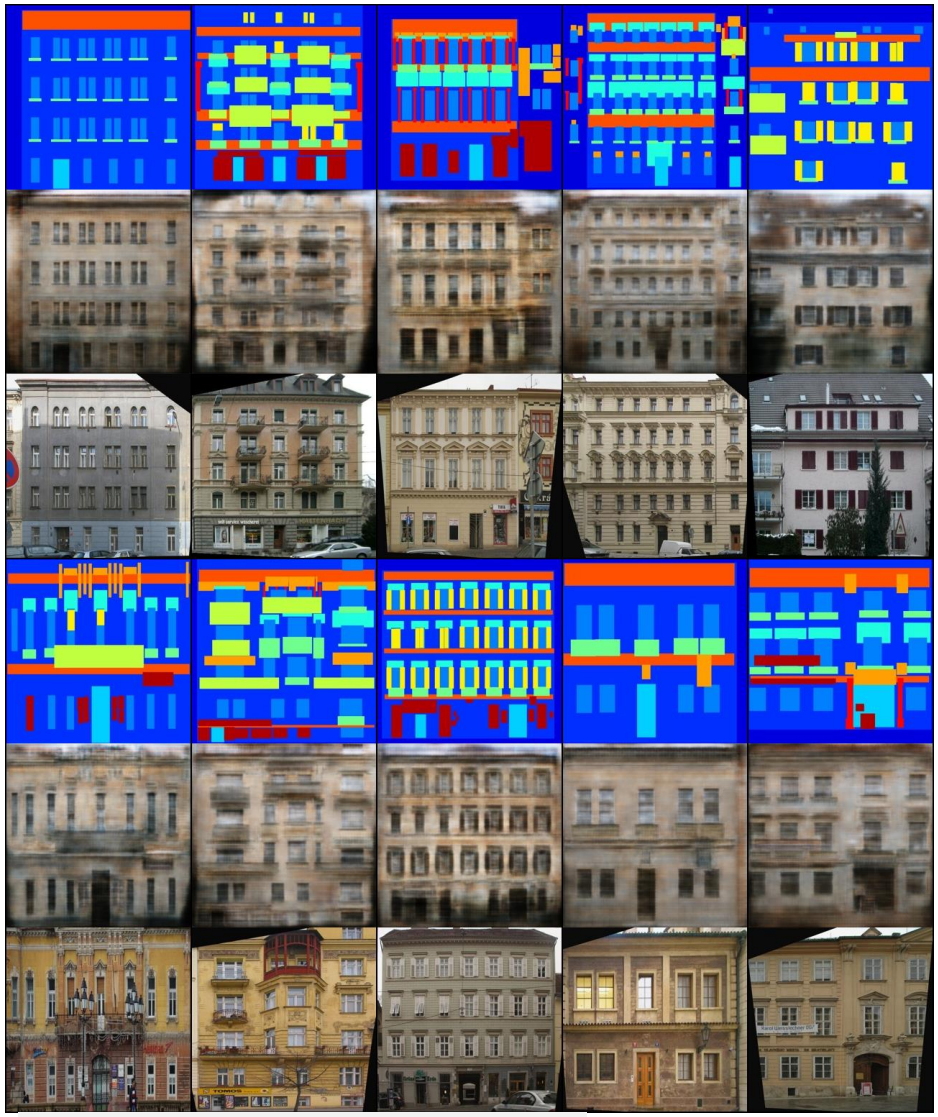
After 24500 iteractions



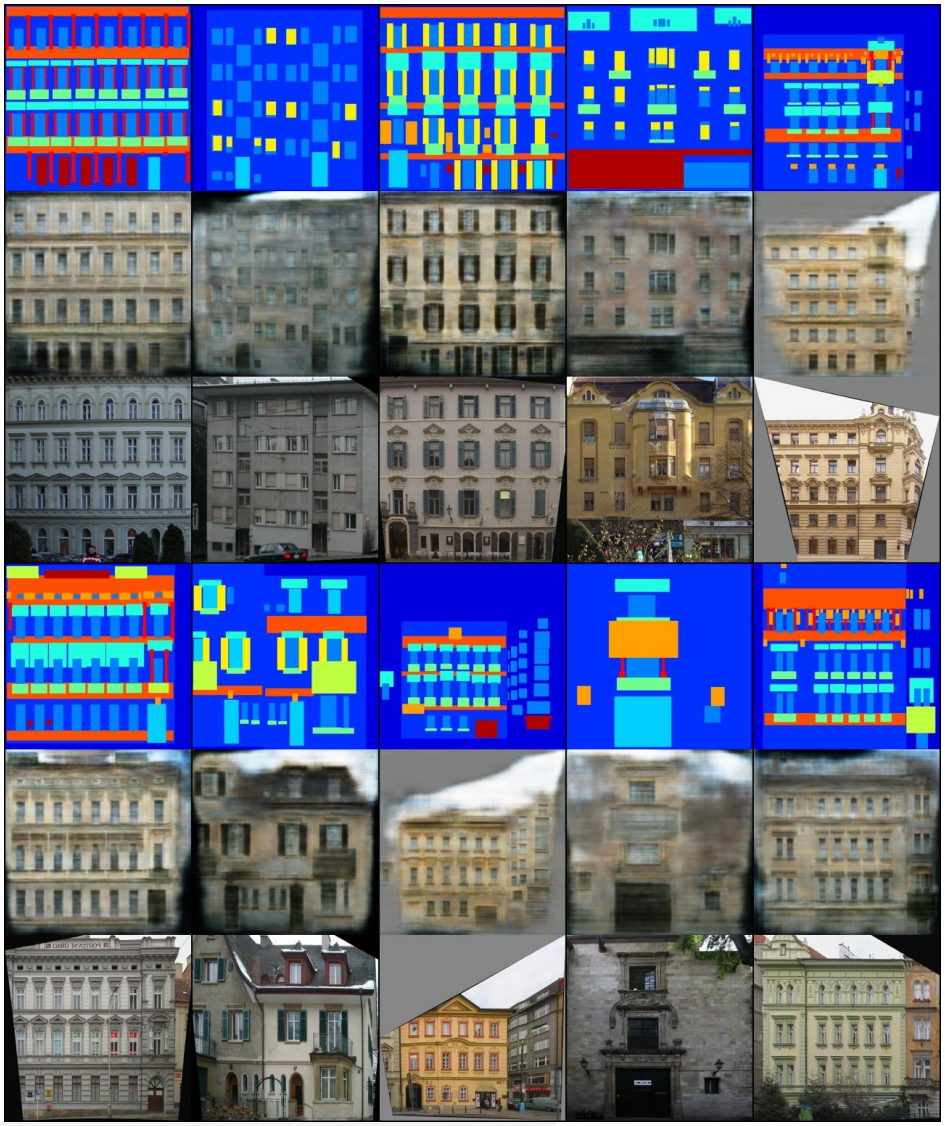
After 36500 iteractions



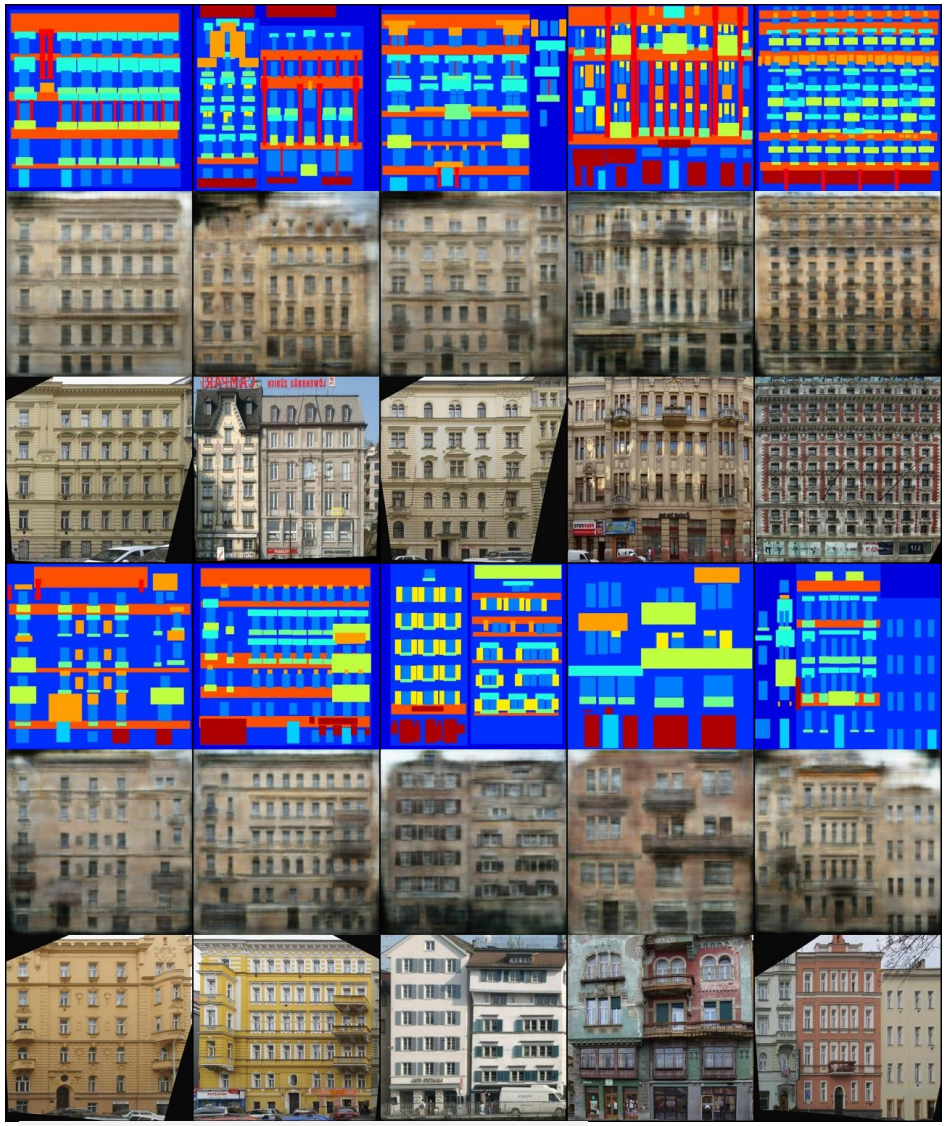
After 48500 iteractions



After 60500 iteractions

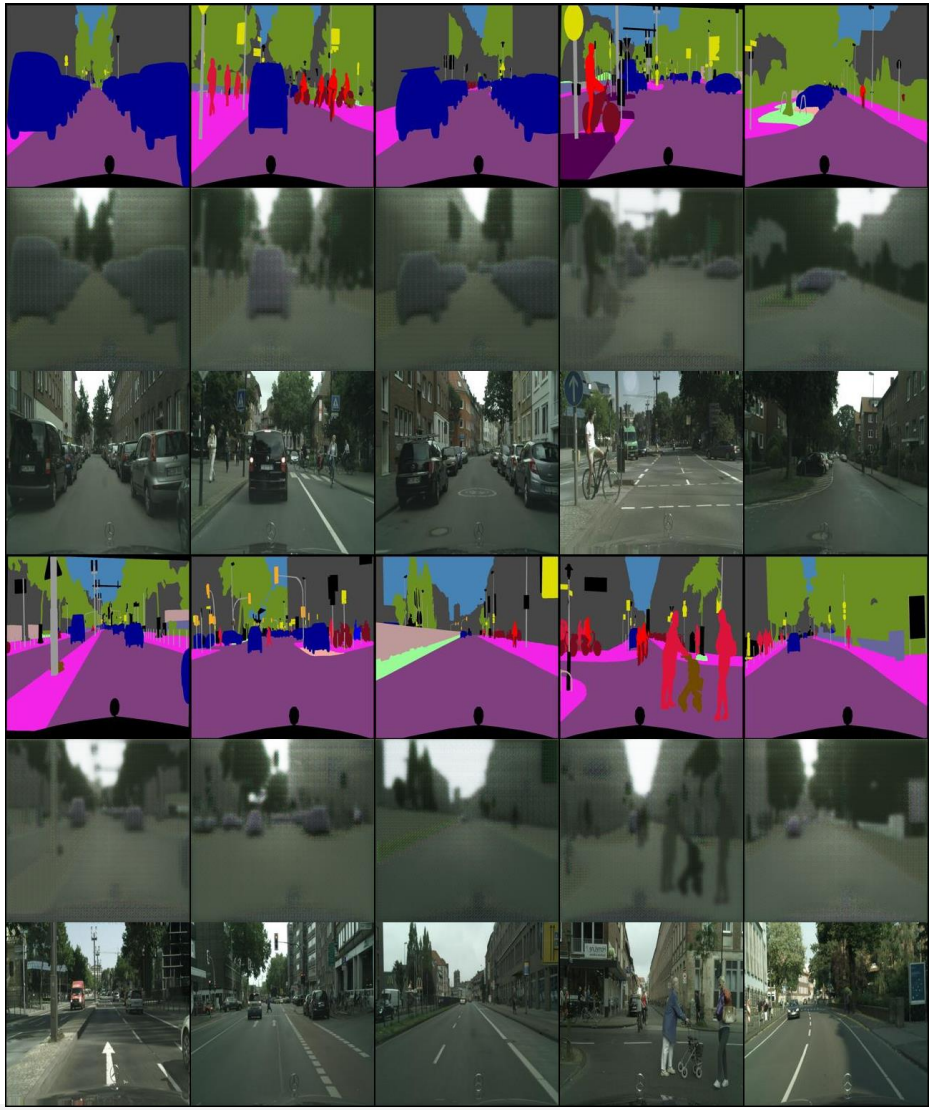


After 72500 iteractions

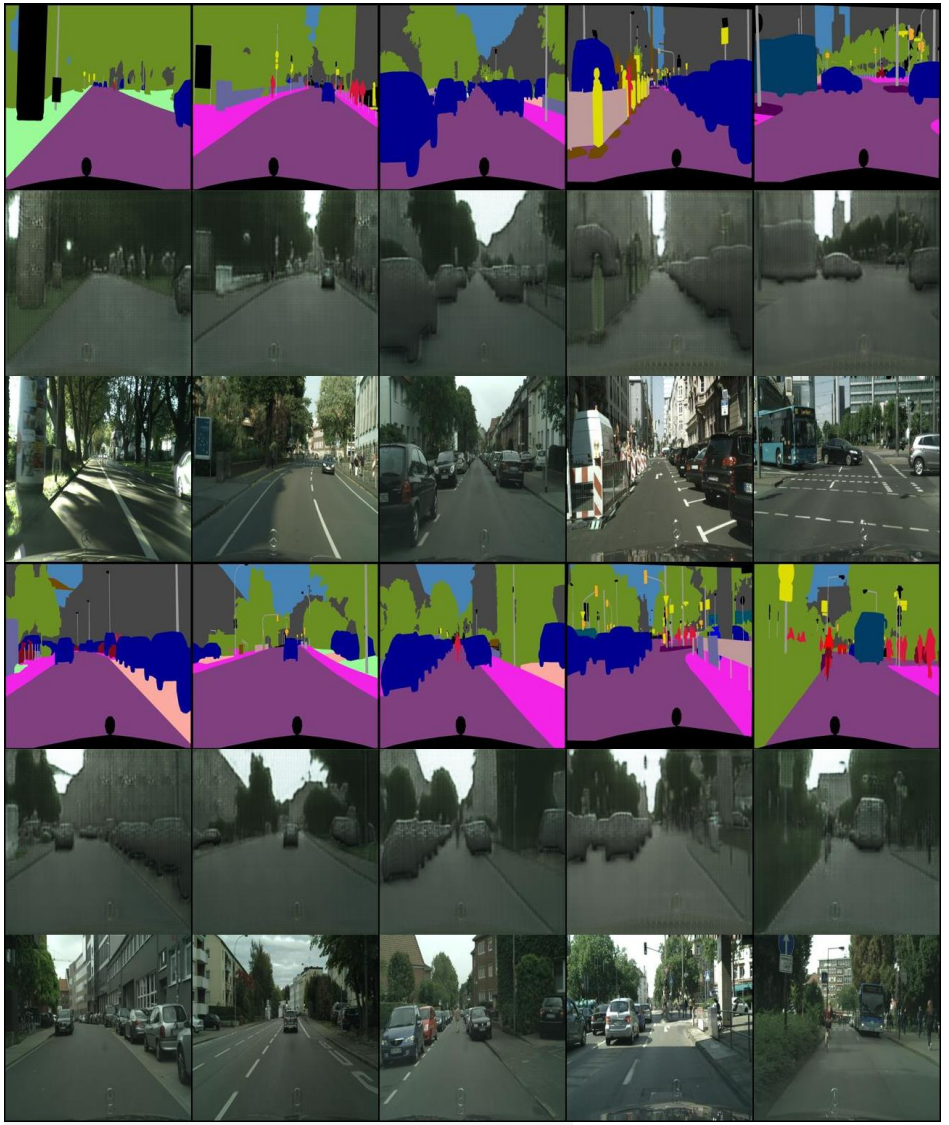


After 101500 iteractions

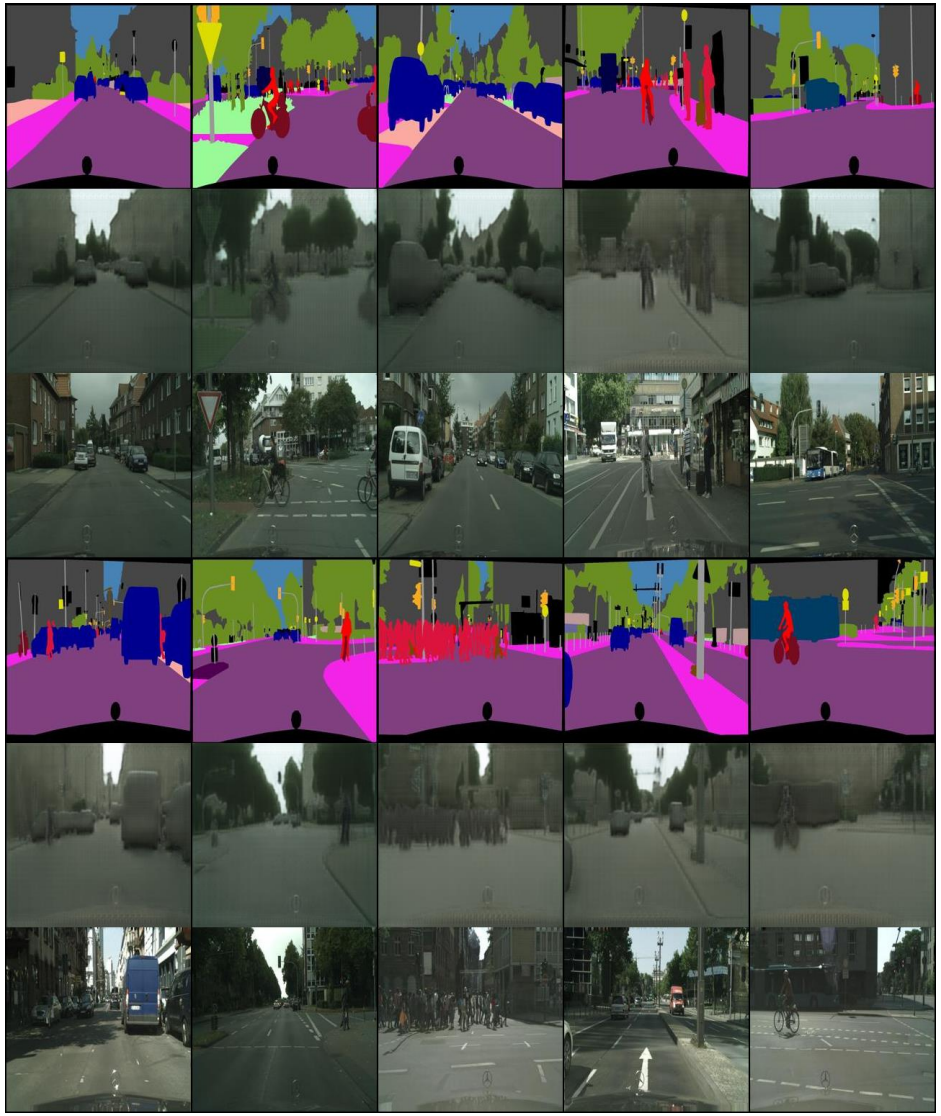
My results on Cityscapes dataset



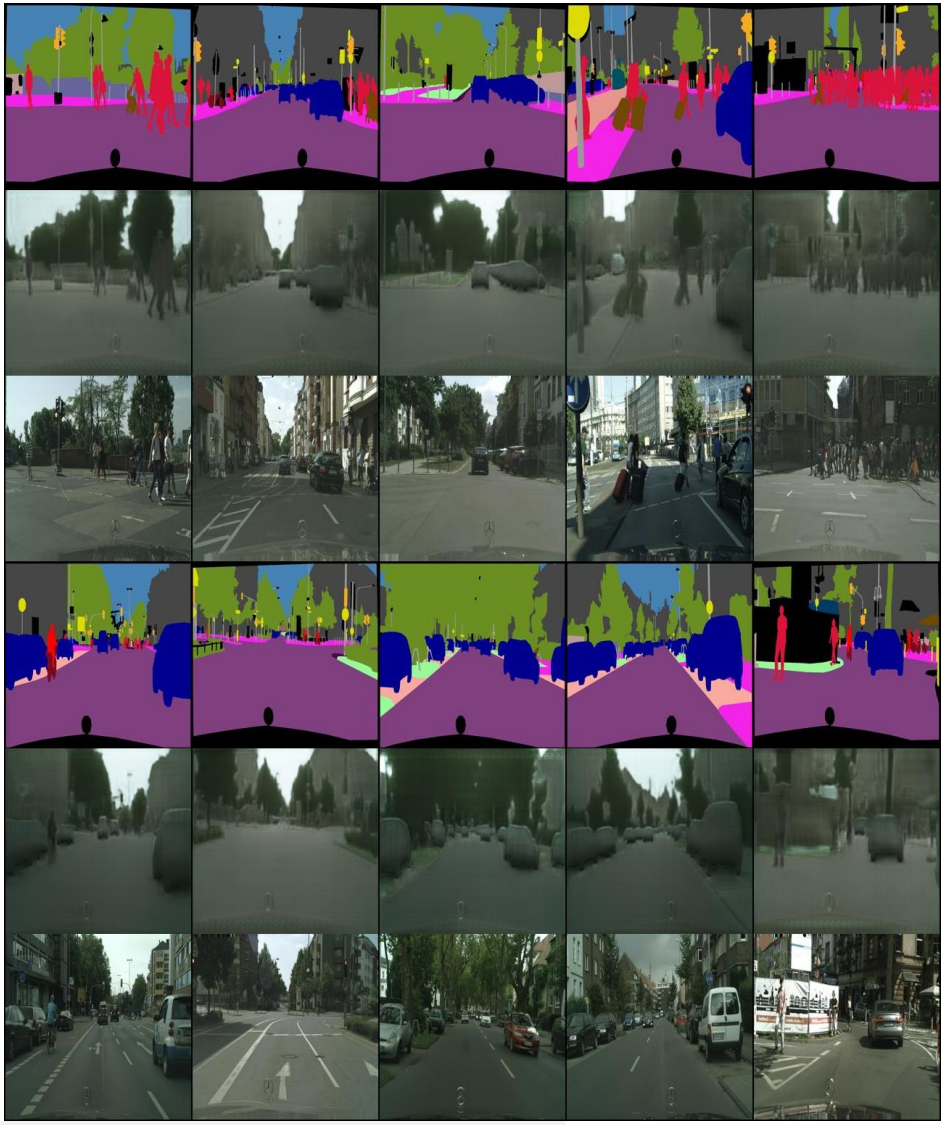
After 500 iteractions



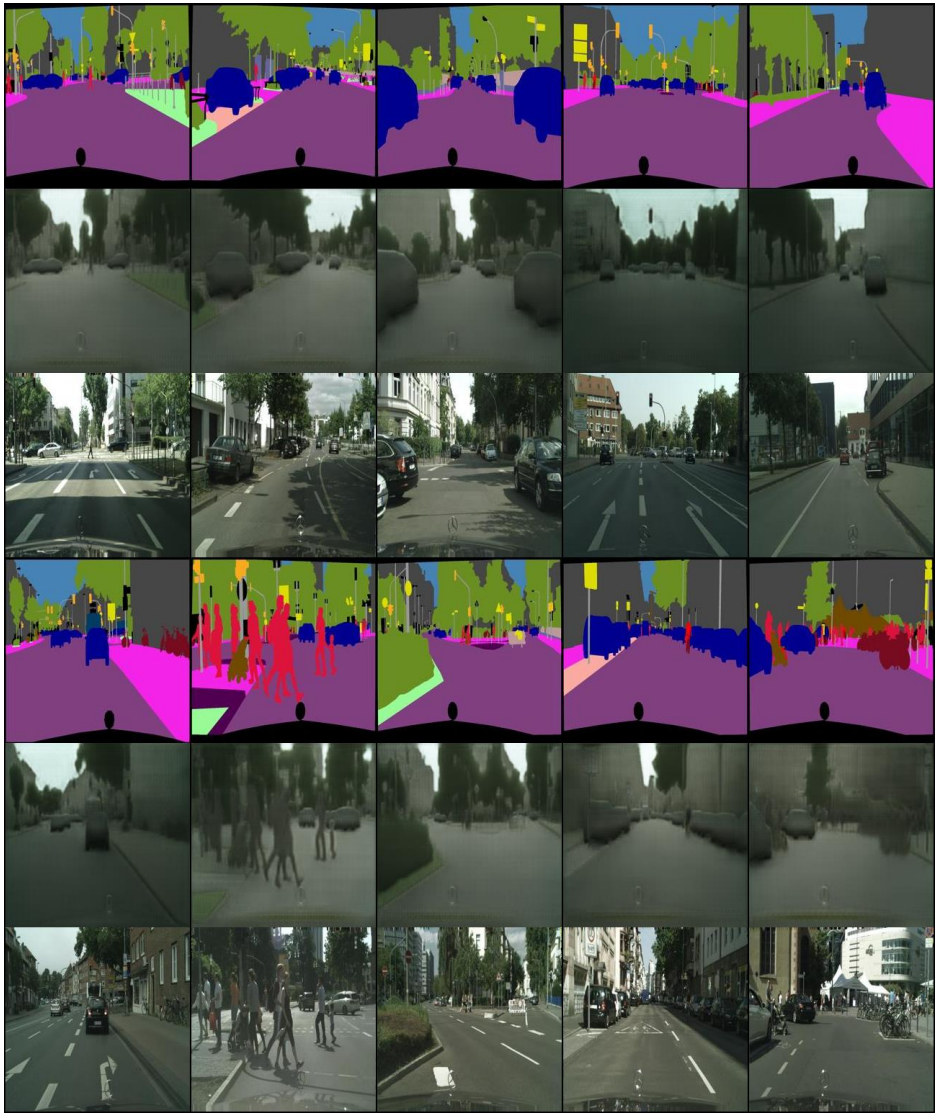
After 12500 iteractions



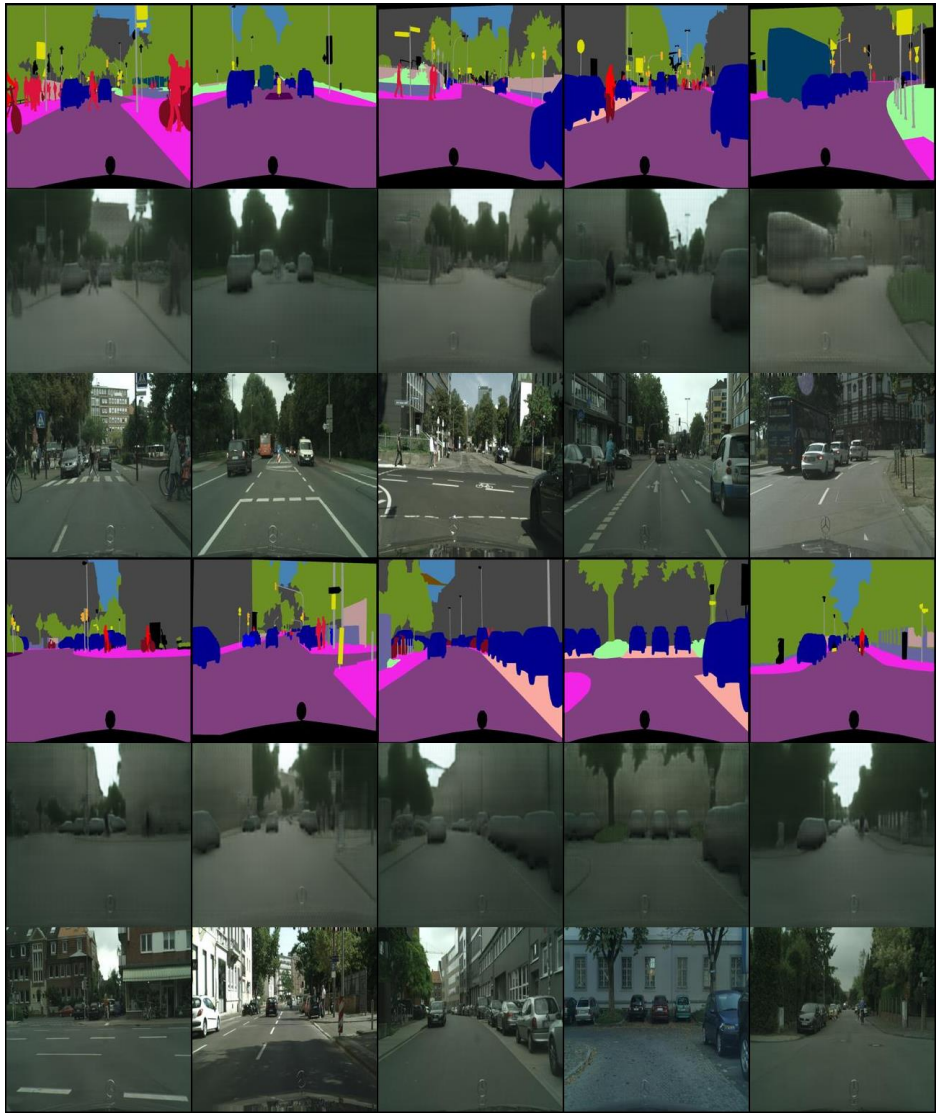
After 25500 iteractions



After 36500 iteractions



After 45500 iteractions



After 59000 iteractions

Thank you